# 160462.1 – Home Screen

**Title of Invention:** Flexible display of information on mobile devices

**Document Author(s):** Neil Enns and Kevin Kennedy

**Introduction**
Every mobile phone on the market has a 'home screen' where the user starts dialing. This screen typically displays the carrier name, battery and signal strength, and provides some mechanism for accessing other functionality on the phone.

We've taken this concept a step further by making the home screen fully customisable by the user. Layouts can be created using an XML-based authoring language that use plug-ins to display information on the page. This information can range from your current carrier name to the latest sports scores to stock quotes. It's very similar to the Digital Dashboard concept from the desktop.

Throughout this document, the following two terms are used heavily. I'm going to define them now in hopes that it will help make the rest of the document and the invention easier to understand.

*Layout:* A text document, written using an XML-based language, that defines the look of a home screen. It consists of sections that include plug-ins, and each section defines the position, fonts, colours, and display of the plug-ins. Advanced users, graphic designers, OEMs, Carriers, etc. can easily author their own layouts.

*Plug-ins:* Typically a .DLL that a software developer has written to work in our home screen framework. Plug-ins display information using the formatting information obtained from the layout. An example of a plug-in is one that displays the current carrier name. Another example is a plug-in that displays a 4-day weather forecast.

**Motivation for the Invention:**
When we were writing the specifications for our smartphone product, we spent a lot of time trying to come up with the one "right" main screen layout that would satisfy the needs of our users. It took us a long time, but we finally realised that there was no "one right" layout. Different people have different needs, and want to see different information when they glance at their phone. Further, a fixed layout does not provide carriers and OEMs with an opportunity to include branded content on the home screen.

The Pocket PC had previously done some work in this area with their Today page. Software developers could write plug-ins and display them on the Today screen of the device. However, this approach meant that only software developers could customise the look of the plug-ins (colours, font sizes, etc.). The average user could not put together a layout and easily change the visual appearance of the plug-ins to suit their tastes.

We solved these problems by creating a home screen with an XML-based language and plug-ins. Instead of having a single home screen, users can now have multiple layout files installed and can switch between the different layouts. To accomplish this we had to invent:

1) The XML-based language for defining the look of an layout
2) The APIs and methods that the plug-ins use to obtain layout information from the XML file
3) A notification scheme so plug-ins can find information out about the state of the device without using too much battery power

**Description of the Invention:**
Since there are several parts to this invention, to simplify things here are descriptions for each part:

### The XML-Based Language

The language is fully documented in the Home Screen specification. It's a fairly simple markup language that can be extended on a per-plug-in basis if needed. Here is a sample of the XML that draws a clock using Arial 10pt in red, using 12-hour format:

```
<plugin file="sysplug.dll" name="clock">
      <time bgcolor="#FFFFFF" fgcolor="#FF0000" font-face="Arial"
font-size="10" mode="12"/>
</plugin>
```

Here is an example that draws the same clock using Times New Roman 12pt in blue, using a 24-hour format:

```
<plugin file="sysplug.dll" name="clock">
      <time bgcolor="#FFFFFF" fgcolor="#0000FF" font-face="Times New Roman"
font-size="12" mode="24"/>
</plugin>
```

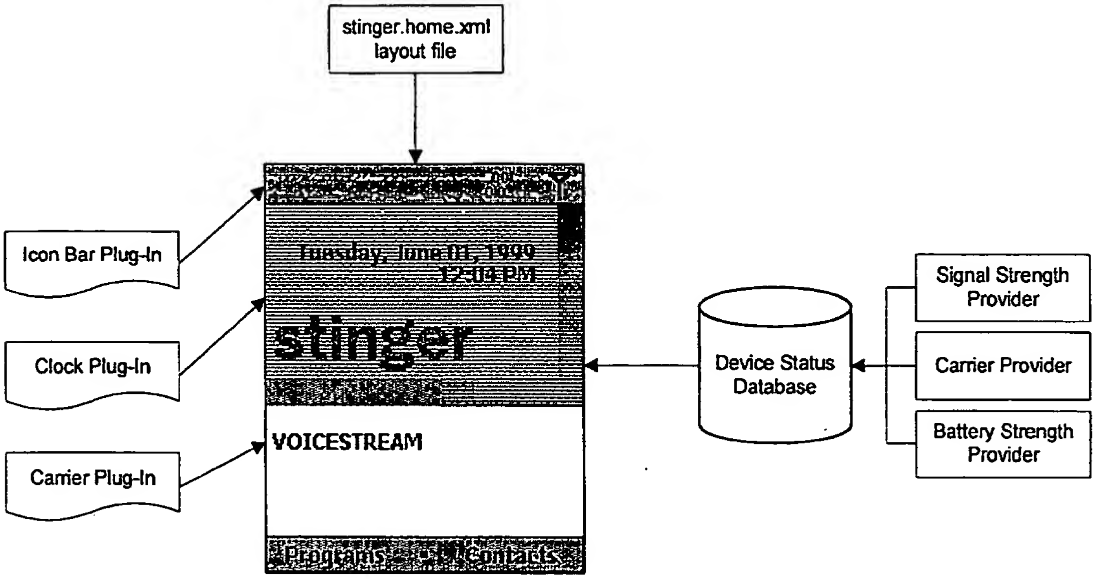
### The Notification Mechanism

The need to display data from disparate applications in the system with high performance is the primary motivator for the current design. Rather than have the individual status producing applications be queried for the data directly by the plug-in, the data is placed by the status producing application in to an intermediate "status store." Status consuming applications that need to display this status in an up-to-date fashion will register themselves with the status store. When the data in the store changes, notifications go out to the registered applications. The registered applications are told which data changed. If they are interested in any of that data, the registered application queries the status store to get the most up to date version. Diagram X shows the flow of updates in the system from data producing applications to data consuming applications.

The home screen is a status consuming application. Each plug-in has specific status elements it is interested in. A "plug-in manager" keeps track of what data each plug-in wants updates for. When the data changes in the status store, the plug-in manager is notified. The plug-in manager runs through its list of plug-ins and causes the ones whose data has changed to redraw. During the redraw, the plug-in requests the updated data from the plug-in manager and displays it. The plug-in manager passes data requests directly to the status store. Diagram X+1 has a UML sequence diagram of this.

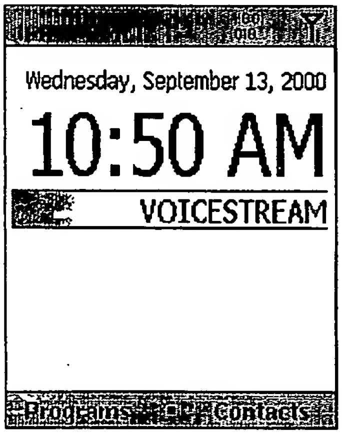
There are many advantages to this mechanism. Accessing data in the store is very fast compared to querying it directly from the owning application. The plug-in uses only a single interface to access all the data it needs to display. The plug-in does not have to cache data that is expensive to acquire, like the phone's signal strength. If data is not immediately available, say at boot when data producing applications are still initializing, the plug-in will be notified when the data is ready. The alternative is for the plug-in to be inactive while it is waiting.

**Diagrams and Flow Charts:**



In the above figure, the home screen reads the stinger.home.xml layout file. The file references three plug-ins that the home screen architecture loads. The plug-ins render their information based on the formatting information in the layout file. Status, such as the carrier name and signal strength, are updated in the device database by external data providers. When the data changes, the plug-ins are notified so they can update their display.

Here is a picture of another home screen that uses the same three plug-ins, but has different formatting options in the XML file:



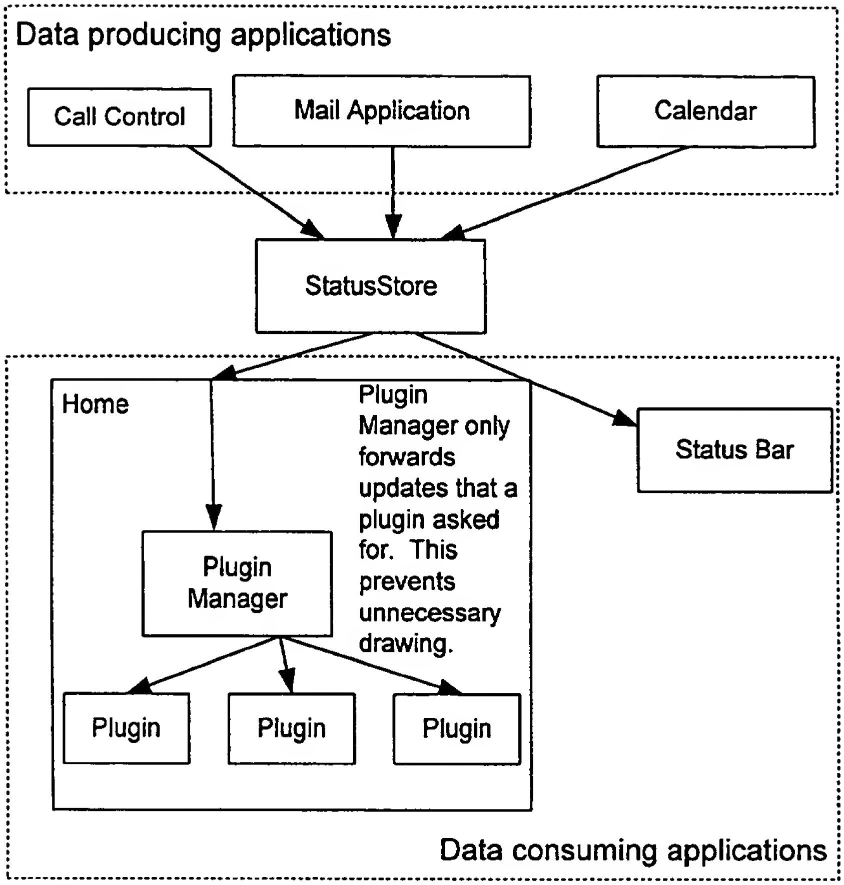

Data producing applications
Call Control
Mail Application
Calendar
StatusStore
Home
Plugin Manager only forwards updates that a plugin asked for. This prevents unnecessary drawing.
Plugin Manager
Plugin
Plugin
Plugin
Status Bar
Data consuming applications

Diagram ☒ – Flow of updates through system

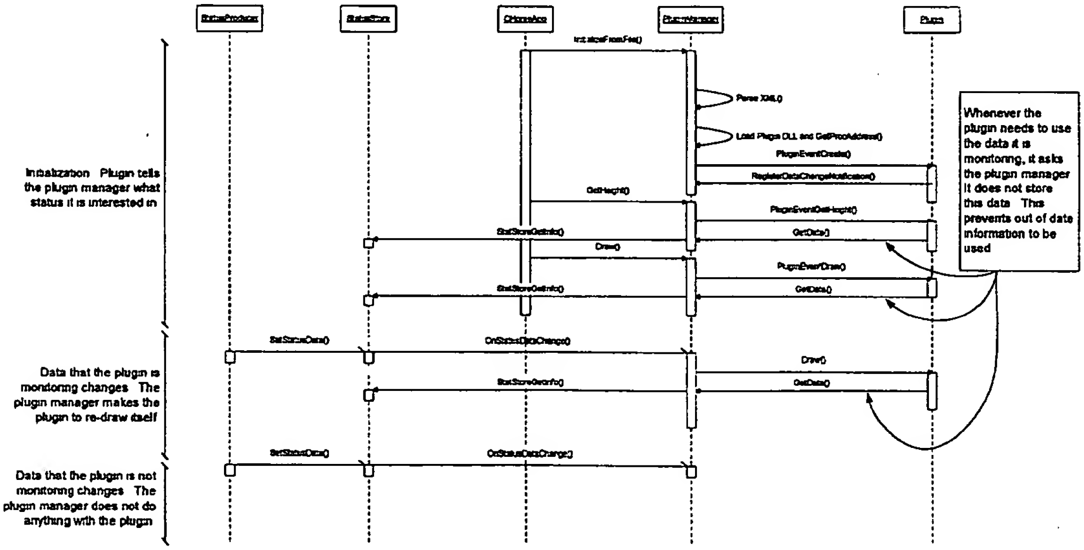

StatusProducer
StatusStore
CHomeApp
PluginManager
Plugin
InitializeFromFile()
Parse XML()
Load Plugin DLL and GetProcAddress()
PluginEventCreate()
RegisterDataChangeNotification()
GetHeight()
PluginEventGetHeight()
StatStoreGetInfo()
GetData()
Draw()
PluginEventDraw()
StatStoreGetInfo()
GetData()
SetStatusData()
OnStatusDataChange()
Draw()
StatStoreGetInfo()
GetData()
SetStatusData()
OnStatusDataChange()
Initialization  Plugin tells the plugin manager what status it is interested in
Data that the plugin is monitoring changes  The plugin manager makes the plugin to re-draw itself
Data that the plugin is not monitoring changes  The plugin manager does not do anything with the plugin
Whenever the plugin needs to use the data it is monitoring, it asks the plugin manager It does not store this data  This prevents out of date information to be used

Diagram X+1 – UML sequence diagram of home screen initialization and status change notification handling.